TradeBlocks

A decentralized token exchange network

Julian Hoang <julian.b.hoang@gmail.com>
Eric Parker <eric.parker13@gmail.com>

1 Changelog

1.1 May 20 (v0.1.0)

- Change token value in examples to use xtb_ addressing

1.2 May 27 (v0.2.0)

- Add section numbering
- Change name of sell action to order action
- Add order execution fees
- Clarify that orders can be used on both the buy and sell side

1.3 June 4 (v0.3.0)

- Change swap blockchain to only contain swap blocks
- Specify offer, commit, and refund actions on the swap blockchain
- Specify read-only fields on swap blockchain

2 Summary

TradeBlocks is a decentralized token exchange network that provides near-instant token trading with high scalability. This is achieved by utilizing a separate blockchain for each account-token pair in the network. To transfer tokens from one account to another, the sender creates a *send* transaction on their own blockchain, and the receiver creates a *receive* transaction on their own blockchain. This makes token transfers asynchronous and massively increases the throughput of the network. To trade one type of token for a different type, an initiator first sends tokens into a swap blockchain and creates an *offer* transaction. Next, a counterparty sends tokens into the swap blockchain and creates a *commit* transaction. Finally, the parties create *receive* transactions on their own account-token blockchains to receive the swapped tokens. If the counterparty doesn't send tokens for the swap, the initiator can create a refund transaction to return their tokens back to their own account-token blockchain. If a fork occurs, the network uses a delegated proof-of-stake protocol to resolve the conflict.

3 Introduction

This document specifies a decentralized token exchange that fulfills the requirements of the Binance Dexathon competition [1]. The competition states that the design of the system must "...choose speed and simplicity over fancy features." In particular, it must implement:

1. Creation, sending, and receiving of tokens
2. Atomic swaps of tokens on the network
3. Trade matching with sell orders

4 Background

The world's first decentralized cryptocurrency, Bitcoin, introduced chaining signed transactions using hashes to provide immutable proof that a sequence of transactions occurred [2]. Users compete to solve a proof-of-work to add a block of transactions to the blockchain every ten minutes. This is a synchronous cryptocurrency because every user must wait for a global event, a block being mined, before new information is added to the network. Only a finite number of transactions can fit in a single block, resulting in an upper limit on the number of transactions that can be processed in a given time period.

Five years after the release of Bitcoin, the Nano (RaiBlocks) whitepaper introduced the concept of an asynchronous cryptocurrency using a directed acyclic graph [3]. In Nano, every account has its own blockchain. Only the account owner can add blocks to their blockchain. To send funds from one account to another, the initiating account creates a *send* transaction on their own blockchain that links to the receiving account. The receiving account then creates a

receive transaction on their own blockchain linking to the originating send transaction. This creates a cross-blockchain link that can verifies, recursively back to the genesis block, that an account owns a certain quantity of funds.

The use of account blockchains and separate send/receive transactions makes the network asynchronous. Users don't wait for a block to be mined before transactions can be added to the network. They simply add it directly to their own blockchain. TradeBlocks builds upon this concept to add token creation, atomic swaps, and sell orders.

5 TradeBlocks Components

5.1 Account

An account is created using a key pair. The public key is the address of the account.

5.2 Transaction

A transaction causes some state in the network to be changed. It doesn't necessarily have to be a transfer of funds. Every transaction must be signed by an account. Each block contains one transaction.

5.3 Blockchains

5.3.1 Account-Token Blockchain

An account-token blockchain contains an immutable sequence of transactions for a single type of token for an account. Only the account owner can add blocks to their own account-token blockchain. An account (public key) may have multiple account-token blockchains associated with it, one for each token type that the account owns.

5.3.2 Swap Blockchain

A swap blockchain atomically swaps tokens of different types from one account to another. It has a specific block sequence, defined later in this document.

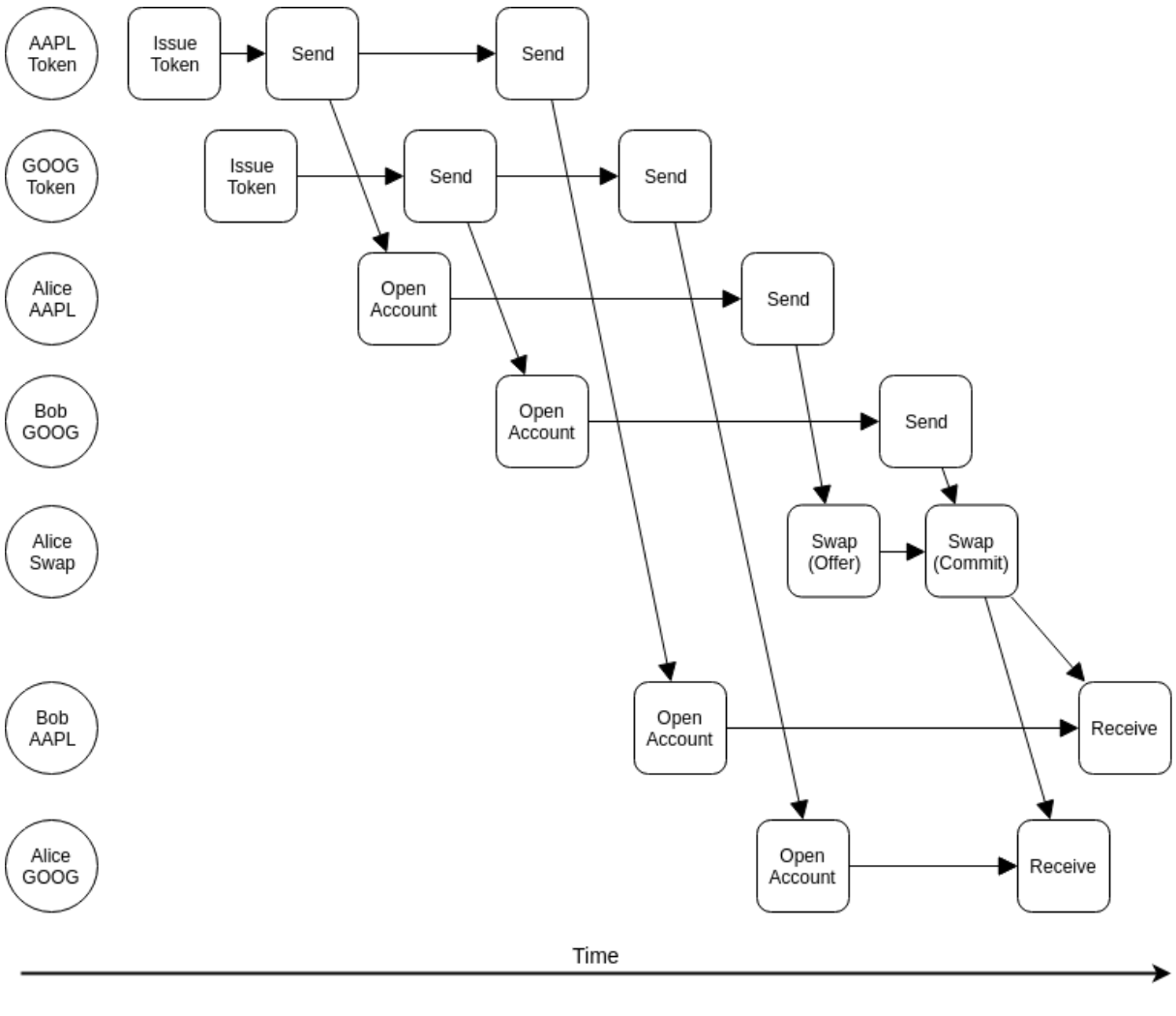
5.3.3 Order Blockchain

An order blockchain contains an account's sell orders for a specific token. To execute an order, a counterparty creates a swap that links to the order.

5.4 Node

A node is a server participating in the TradeBlocks network. It services requests from users and manages the accounts that the node controls.

6 System Overview



Example scenario: Alice and Bob receive AAPL and GOOG tokens. Alice wants to swap some of her AAPL for GOOG with Bob. She creates a swap blockchain and sends some AAPL to it. Bob accepts the swap by sending GOOG to it. Alice then receives GOOG from the swap and Bob receives AAPL.

6.1 Creating an Account

To create an account, the user first creates a public key pair. Then, another account on the network creates a *send* transaction with the *link* field set to this public key. Finally, the user

creates an *open* transaction with the *link* field set to the hash of the originating send transaction. This receives the tokens, shown in the *balance* field, and designates the start of an account-token blockchain.

```
{
  “action”: “open”,
  “account”: “A9CA21C23A…”,
  “token”: “xtb_ba15…”,
  “previous”: null,
  “representative”: “xtb_ba31…”,
  “balance”: 100,
  “link”: “B1A8ECA21…”
}
```

Example of an open transaction

6.2 Creating a Token

Tokens are generated by creating an *issue* transaction as the first transaction of an account, as opposed to an *open* transaction. The *issue* transaction sets the total quantity of tokens in its *balance* field. The quantity of tokens cannot be increased after the token is created. It is not permitted to have multiple issue transactions in a single account-token blockchain.

```
{
  “action”: “issue”,
  “account”: “xtb_ba15…”,
  “token”: “xtb_ba15…”,
  “previous”: null,
  “representative”: “xtb_ba31…”,
  “balance”: 24000000,
  “link”: null
}
```

Example of an issue transaction

The public key of the account becomes the global identifier for the token. To distribute this token to other accounts, the token owner creates *send* transactions to the destination accounts.

6.3 Account Balance

The current balance of an account is recorded in every block as the *balance* field. To verify an account’s balance, the verifier checks every linked block leading up to that balance, which will eventually go back to the originating *issue* transaction that created the token.

6.4 Sending From an Account

To send tokens from one account to another, the user creates a *send* transaction that specifies the receiving account in the *link* field. The amount sent is implicitly determined by the subtraction of the previous block's balance from the send block's balance. Once the *send* transaction is created, it cannot be revoked, and the recipient is entitled to receive the tokens using a *receive* transaction.

```
{
  "action": "send",
  "account": "C604354B1…",
  "token": "xtb_ad81…",
  "previous": "1967EA355…",
  "representative": "xtb_ba31…",
  "balance": 23999900,
  "link": "xtb_3wam…"
}
```

Example of a send transaction

6.5 Receiving a Send

To claim tokens that were sent to an account, the receiver creates a *receive* transaction that specifies the originating send transaction in the *link* field. The amount received is implicitly determined by the subtraction of the receive block's balance from the previous block's balance. This amount can be verified by checking the linked transaction to see the quantity of tokens that the sender sent.

```
{
  "action": "receive",
  "account": "211E2B3F7C…",
  "token": "xtb_ad81…",
  "previous": "DC04354B1…",
  "representative": "xtb_ba31…",
  "balance": 200,
  "link": "DC9A2B3C4E…"
}
```

Example of a receive transaction

6.6 Swapping Tokens

To swap tokens of differing types with another account (counterparty), the initiator sends tokens to an intermediate swap blockchain. Every block in a swap blockchain has an *id* field, chosen by the initiator, that specifies the unique id of the swap. The address of a swap is its

account-id pair. A swap blockchain is the only blockchain that allows different accounts to add blocks to the same chain. It has a specific transaction order described below, and any transaction that doesn't conform to the order is not allowed.

The first block, known as the *offer* transaction, specifies in the *left* field the originating *send* transaction, representing the initiator's portion of the swap. The *counterparty* field specifies the account that will receive the tokens sent. The initiator specifies in the *want* and *quantity* fields the type and quantity of tokens they want in return.

```
{
  "action": "offer",
  "account": "xtb_1ba0…",
  "token": "xtb_ad81…",
  "id": "ea214567…",
  "previous": null,
  "left": "0A2B3E6C…",
  "right": null,
  "refund_left": null,
  "refund_right": null,
  "counterparty": "xtb_3wam…",
  "want": "xtb_ba15…",
  "quantity": 50,
  "executor": null,
  "fee": null
}
```

Example of an offering swap transaction

Since the initiator must send tokens to the swap blockchain before creating the *offer* transaction (which specifies the originating send transaction in the *left* field), the swap is "pre-loaded" and waits for the counterparty to send their tokens to commit the swap.

The counterparty accepts the swap by sending the requested tokens to the address of the swap (the account-id pair). Then, they create a *commit* transaction that specifies the offer in the *previous* field and the counterparty's originating send in the *right* field. All of the other fields must match the parent swap. The *send* transaction in the *right* field must contain the tokens specified in the *want* and *quantity* fields. This commits the swap and allows both accounts to receive their respective tokens using a *receive* transaction on their account-token blockchain that links to this commit swap.

Action	Signer	Pos.	Input	Output
offer	account	1	initiator *send*	
commit	counterparty/executor	2a	counterparty *send*	initiator *receive* and counterparty *receive*
refund-left	account	2b		initiator *receive*
refund-right	counterparty/executor	3b	counterparty *send*	counterparty *receive*

All actions on the swap blockchain (v0.3.0)

6.6.1 Cancelling a Swap (v0.3.0)

It's possible that the counterparty doesn't send tokens into the swap. The initiator can get their tokens back by refunding themselves at any time before the counterparty creates a commit swap (if ever). This is done by creating a *refund-left* transaction on the swap blockchain that specifies the offer in the *previous* field and the initiator's account address in the *refund_left* field. The initiator can then create a *receive* transaction on their own account-token blockchain linking to the refund transaction. Once the *refund-left* transaction is created, the swap is considered to be cancelled.

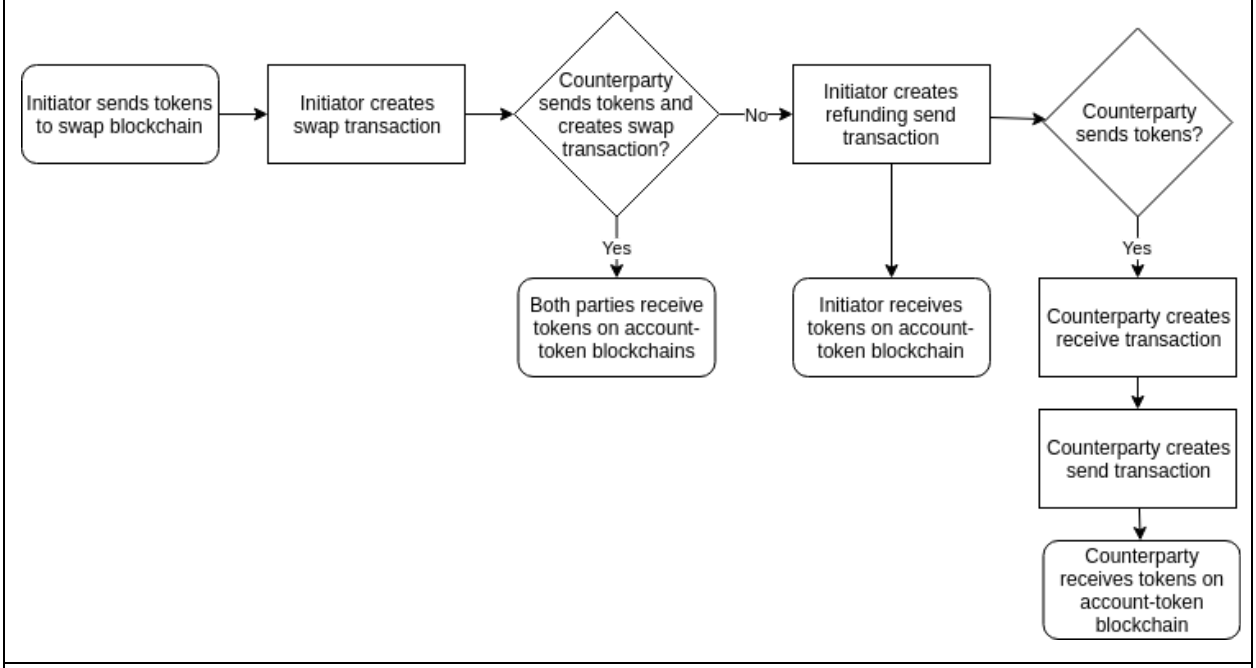
There exists a potential race condition if the initiator creates a *refund-left* transaction at the same time the counterparty creates the *commit* transaction. In this case, there will be two transactions that claim the same block as their parent. The network detects this conflict and uses a delegated proof-of-stake protocol to choose which transaction to keep. If the *commit* transaction survives, the parties proceed as normal to receive their swapped tokens.

If the initiator's *refund-left* transaction is chosen as the surviving block, the counterparty needs to retrieve their own tokens that they sent to the swap blockchain. They do this by creating a *refund-right* transaction with the *previous* field set to the refund-left block, the originating send in the *right* field, and their own account address in the *refund_right* field. Then, they can retrieve their tokens by creating a *receive* transaction on their account-token blockchain.

The target address of a refund must match the source address where the tokens originally came from. This may be an account-token address, or an account-id address if the tokens were originally sent from an order.

6.6.2 Final State of a Swap

Due to the rules specified above, there exists a specific order that transactions must appear in the swap blockchain. This results in one of three final states: either the tokens are swapped, the initiator gets refunded, or both the initiator and counterparty get refunded.



The possible outcomes of a token swap attempt (v0.1.0, out of date)

6.6.3 Swap Executor (v0.2.0)

The optional *executor* represents the account that will receive the quantity of tokens in *fee* once the swap is committed. This is intended for order execution fees, specified in 6.7.1.1. If a fee is specified, the initiator must send a quantity of tokens equal to the *quantity* plus *fee* into the swap. The executor receives the fee by creating a *receive* transaction on their account-token blockchain with the *link* set to the commit swap.

6.6.3.1 Executor Signing on Behalf of the Account Owner (v0.3.0)

The executor may sign the *commit* or *refund-right* transactions on behalf of the account owner if and only if they are also specified as the executor in the order that matches the account-swapid pair, if any.

6.6.4 Read-Only Fields (v0.3.0)

On the swap blockchain, any field that is not null must match the value in the previous block, with the sole exception of the *action* and *previous* field itself. This ensures that the final

state of the swap is effectively cached in the *commit* transaction, or in the *refund-left* and *refund-right* transactions.

6.7 Creating an Order

To create a buy or sell order, the user first creates a new order blockchain, then sends the base token to it, and finally creates an *order* transaction. The initial *order* transaction has the originating *send* transaction in the *link* field. The *quote* and *price* fields indicate the type of token and the price per unit they are willing to trade for. The *partial* field indicates if this seller is willing to accept partial fills.

(v0.2.0) An order transaction can be used to represent either the buy or sell side of a traditional exchange. For example, to create a sell order for a pair with a quote currency of QUOTE and base currency of BASE, the user would create an order transaction with *token* set to BASE and *quote* set to QUOTE. To create a buy order for this pair, the user would create a "reverse" order with *token* set to QUOTE and *quote* set to BASE.

```
{
  "action": "create-order|accept-order|refund-order",
  "account": "xtb_8ce10a…",
  "token": "xtb_4a11b7…",
  "id": "391AC31E…",
  "previous": null,
  "balance": 30,
  "quote": "xtb_1a9b21c…",
  "price": 2
  "link": "5121B2CC…",
  "partial": true,
  "executor": "xtb_b1na7ce…",
  "fee": 0.03
}
```

Example of an order transaction

6.7.1 Executing an Order

To execute an order, a taker creates an *offer* transaction on a swap blockchain that has the *counterparty* field set to the maker's account and the *id* field set to the same value as the order. The maker can than accept the order by creating an *accept-order* transaction with the *link* field set to the address of the swap, decrementing the *balance* to specify the quantity of tokens sent, and creating a *commit* transaction on the swap blockchain with the *right* field set to the accept-order. The traded tokens can then be withdrawn from the swap blockchain as in an ordinary swap.

(v0.2.0) The quantity of tokens in the offering *swap* transaction must match the rate in the *price* of the *order* transaction. It must supply the total *balance* in the order, unless *partial* is true.

6.7.1.1 Execution Fee (v0.2.0)

While the maker can receive *offer* swap transactions while offline, they must be online at some point in order to finalize the swap by publishing the *commit* swap transaction. To relieve the burden of having to be online, the maker can specify an *executor* that will publish the required *commit* swap transaction in exchange for the specified *fee*. The executor may be a cryptocurrency exchange with reliable nodes that are always online.

The fee represents the total quantity of *quote* tokens that the executor can receive after publishing the *commit* transaction. The executor cannot receive tokens from the *order* transaction directly. The fee rate is determined by off-chain information, such as an HTTP request to the executor's web server. If the maker specifies an incorrect *fee* in the *order* transaction, the executor can simply refuse to publish the final *commit* transaction.

If an executor is specified, it is authorized to sign the following transactions on the corresponding order and swap blockchain. This allows an account holder to use an executor without revealing their private key. The executor signs the following transactions on behalf of the account holder:

1. On the order blockchain, a sending *order* transaction that matches a valid *offer*
2. On the swap blockchain, the *commit* transaction required to finalize the trade
3. On the swap blockchain, the *refund-right* transaction if the swap was cancelled

6.7.2 Handling a Cancelled Swap

If the taker ends up refunding their tokens in the swap, it may be necessary for the maker to also refund their tokens back into their order blockchain. They do this by creating an *order* transaction that has the *link* field set to the *refund-right* transaction on the swap blockchain, and incrementing the order *balance* by the amount being refunded.

6.7.3 Cancelling an Order

To cancel an order, the you create an *order* transaction with the *link* field set to your own account-token blockchain and set the *balance* field to zero. Then, create a *receive* transaction on your own account-token blockchain to receive your tokens.

6.8 Assigning a Representative

The network uses a delegated proof-of-stake protocol to resolve a conflict if two or more valid blocks are created that specify the same parent. Proof-of-stake requires that the node be

online and actively participating in the network. To relieve themselves the responsibility of being online, an account can use a *change* transaction to specify that a different account will represent their share of the vote. An account can also choose to represent itself. Representatives are only used for proof-of-stake voting and cannot send the account's tokens.

```
{
  "action": "change",
  "account": "1312B3F7C…",
  "token": "BA1F1E0F…",
  "previous": "93CAA05B3…",
  "representative": "xtb_ba31…",
  "balance": 200,
  "link": null
}
```

Example of an change transaction

6.9 Forks and Voting

A fork occurs if an account tries to sign two transactions that specify the same parent (double spend attempt). Additionally, a fork occurs when the initiator of a swap creates a send and the counterparty creates a commit swap at the same time. In either case, the network must choose one transaction to keep and the other to discard using the proof-of-stake protocol.

When a node detects a fork, it will broadcast a vote for the transaction that it saw first. The weight of its vote is the sum of the quantity of tokens in accounts that have the node specified as their representative. In the event of a send-commit fork on a swap blockchain, the token counted is the token of the swap initiator.

Each node will keep a tally of the votes from each representative. Every minute, for four minutes, the nodes will switch their vote to whichever transaction is winning. This makes it likely that the network will come to a consensus on which transaction to keep, since the winning transaction increases their vote margin with every round of voting.

6.10 Proof of Work

Since there are no fees to create transactions, a proof-of-work requirement is added to all transactions in order to reduce spam. Every type of transaction has an additional *work* field that must be set with a correct value for the transaction to be verified. The transaction signer must create a string consisting of a nonce that is appended to the *previous* field, or the *account* field if there is no previous block. The computed hash of this string must then be under a certain threshold value.

The proof-of-work is only intended to combat spam, and isn't used for consensus or conflict resolution. As a result, it should only take a few seconds or less to calculate. Clients can precache proof-of-work solutions because they already know the *previous* field of the current block. This allows for an instant transaction when the proof-of-work is precached.

7 Attack Vectors

As TradeBlocks builds upon the block graph concept of Nano, it is also subject to the same attack vectors as Nano. These include block gap synchronization, transaction flooding, sybil attack, penny-spend attack, precomputed proof-of-work attack, >50% attack, and bootstrap poisoning. See the Nano whitepaper for details and mitigation techniques for these attacks [3]. Below, only attack vectors specific to TradeBlocks will be discussed.

7.1 False Sell Orders

There exists no runtime requirement that sellers actually accept incoming swaps for their sell orders. As a result, in an attempt to manipulate the market price, sellers could create large quantities of orders at low prices without ever intending to accept swaps for them. To mitigate this attack, clients should check how many swaps are pending or cancelled for an order (by matching the swap id with the order id). If no swap has been committed, and a statistically significant number of swaps are pending or cancelled, this indicates that the seller isn't fulfilling their orders and should be ignored.

7.2 False Swaps

A counterparty could repeatedly create swaps for a sell order but then cancel the swap before the seller has a chance to broadcast the commit swap. While risky due to the chance that the network will vote for the commit swap when resolving the conflict, it could cause the seller's tokens to be endlessly locked up in the attacker's swap blockchains. While the attacker cannot spend the tokens, this prevents legitimate buyers from filling the sell order, effectively creating a denial of service attack. To mitigate this attack, sellers should not transact with buyers that have cancelled swaps against them in the past.

7.3 Token Flooding

Since there is no fee to create a new token, an attacker could create a large number of different spam tokens and send them to other accounts to create unclaimed token balances. In a naive client UI, this could fill up the user's screen with a large number of useless tokens that were sent to their account. To mitigate this attack, clients should only display tokens that the user explicitly wishes to receive.

8 Conclusion

TradeBlocks builds upon the block graph concept of Nano to add token creation, atomic swaps, and trading functionality. Tokens can be created by anyone who wishes to do so and traded without requiring a central exchange. The simple network design eliminates global state, improving performance and providing near-instant transaction capability.

9 References

[1] https://medium.com/binanceexchange/binance-dexathon-845dc0cbfffe
[2] https://bitcoin.org/bitcoin.pdf
[3] https://nano.org/en/whitepaper